Санкт-Петербургский государственный университет

Кафедра технологии программирования

Юревич Владислав Сергеевич

Выпускная квалификационная работа бакалавра

Проектирование и разработка системы автоматизированной медицинской диагностики (на примере диагностики заболевания Паркинсона)

Направление 010400.62

Прикладная математика и информатика

Научный руководитель,
доктор технических наук,
профессор
Печников А. А.

Санкт-Петербург

2017

Содержание

Введение

В настоящее время сложно переоценить роль искусственного интеллекта в современном мире. Изобилие форм и областей применения демонстрирует силу и огромные возможности машинного обучения: от роботов, побеждающих международных гроссмейстеров в шахматы, до автопилотируемых девайсов и автоматизированных игроков на фондовых рынках.

Большое количество исследований в научном мире, связанных с машинным обучением, ведется в области здравоохранения. Вопросы медицины вряд ли когда-либо потеряют свою важность, так как одной из основных целей современного общества является улучшение качества жизни, что тесно связано со здоровьем населения планеты. Именно поэтому актуально проводить исследования в области машинного обучения, связанные медициной, и разрабатывать программы и алгоритмы, способствующие развитию здравоохранения.

Разработанные инструменты медицинской диагностики не могут заменить всю работу врача. Это было бы рискованно и неэтично полагаться на пусть даже хорошо разработанный искусственный интеллект. Однако подобные программы могут существенно помочь доктору и ускорить зачастую длительный процесс диагностирования пациентов. Применение подобных инструментов необходимо, когда постановка диагноза усложняется большим количеством медицинских данных или редким заболеванием.

В данной выпускной квалификационной работе создавалась система автоматизированной медицинской диагностики. Ее работа демонстрируется на примере анализа голоса и речи пациентов с подозрением на заболевание Паркинсона. В настоящее время данная болезнь неизлечима и сложно диагностируется на ранних этапах. Поэтому для демонстрации работы системы была выбрана именно она.

С использованием современных технологий было разработано веб-

приложение, с помощью которого лаборант, принимающий анализы у пациентов, мог бы пользоваться системой, получая результаты анализов в короткие сроки.

Постановка задачи

В ВКР были выделены несколько основных задач:

1. С помощью открытой библиотеки Scikit-learn [1] на языке Python протестировать такие алгоритмы машинного обучения, как метод опорных векторов, метод дерева решений, метод k наиближайших соседей;
2. На языке программирования Java реализовать многослойную нейронную сеть;
3. Сравнить эффективность алгоритмов с помощью пятиэтапного скользящего контроля;
4. Разработать веб-приложение, демонстрирующее работу системы автоматизированной медицинской диагностики.

Обзор литературы

При написании данной работы были использованы научная, а также учебно-методическая литература, публикации в различных научных изданиях и интернет-ресурсы.

В исследовании принимаются во внимание такие методы машинного обучения как метод опорных векторов, метод дерева решений, метод k-ближайших соседей. Их выбор основан на статье [2], где подробно описывается суть алгоритмов, их преимущества и недостатки. В книге [3] рассматриваются устройство и работа нейронных сетей. Особый интерес представляет глава 5 параграф 3, где описывается алгоритм обратного распространения ошибки. Именно этот метод используется в данной работе для обучения многослойной нейронной сети.

Для сравнения эффективности работы методов машинного обучения были использованы метрики оценки качества классификации, о которых написано в статье [4]. Не менее важной является статья [5], где уделяется больше внимания оценке классификации, когда объект может принадлежать только одному классу (single-label classification), что соответствует задаче, описываемой в данной работе. Для более объективного сравнения эффективности работы алгоритмов использовался метод скользящего контроля, который описывается в работе [6].

При разработке веб-приложения особое внимание нужно уделить проектированию его архитектуры. О различных ее шаблонах и критериях сказано в книге [7]. При создании приложения использовались современные инструменты разработки, упрощающие работу программиста. Они помогают избежать использование повторяющегося, "закулисного"кода, тем самым ускоряя процесс создания готового продукта. О Spring framework подробным образом написано в книге [8]. Авторы тщательно рассматривают изобилие возможностей фреймворка, способы его настройки и конфи-

гурации. Для того, чтобы установить связи между объектами приложения и элементами таблиц базы данных использовалась такая ORM, как Hibernate. Об особенностях ее работы написано в книге [9].

Глава 1. Описание медицинских данных

2.1. Описание болезни Паркинсона

Болезнь Паркинсона (Идиопатический синдром паркинсонизма) — неврологическое заболевание, характеризующееся дегенерацией моторной системы. Тремор конечностей, неспособность удерживать мышцы в тонусе, снижение двигательной активности (гипокнезия) и мышечное несопротивление к деформации — основные симптомы этого недуга. Идиопатический синдром паркинсонизма — хроническое и медленно прогрессирующее заболевание.

Причиной этой болезни является разрушение и гибель нейронов, вырабатывающих дофамин в черной субстанции центральной нервной системы. Черная субстанция имеет особое значение в статокинетической функции, регуляции моторной функции и тонуса мышц. Благодаря ей осуществляются процессы движения глаз, дыхания, жевания и глотания.

По словам невропатолога Александра Пянктовского, болезнью Паркинсона страдают около 0,1% населения страны. Из них подавляющее большинство (>80%) — люди старше 60 лет. Наблюдения недавних лет показывают, что ситуация ухудшается: количество больных растет, а средний возраст уменьшается.

В изучении этого заболевания принимают участие ученые со всего мира. На данный момент болезнь Паркинсона неизлечима. Более того, она сложно диагностируется на ранних этапах развития. А чем позже обнаружить заболевание и начать лечение, тем тяжелее будут последствия. По этой причине к исследованию симптоматики приковано внимание многих научных деятелей в области медицины.

2.2. Описание анализов голоса и речи

Существует большое количество методов диагностики заболевания Паркинсона. Самый простой и часто используемый из них – это обследование физического здоровья человека: достаточно наличия гипокнезии и одного из симптомов, описанных выше. Также на ранних этапах, когда симптомы проявляются слабо, прибегают к выявлению постуральных рефлексов (рефлексы положения). Выявить паркинсонизм помогает дифференциальный диагноз, в ходе которого исключаются состояния и процессы, имеющие симптомы, похожие на болезнь Паркинсона.

В данном исследовании за основу взяты результаты речевого и голосового анализа, проведенного в Турции учными различных институтов [10]. Результаты получены из открытого репозитория данных для машинного обучения [11]. В обследовании принимают участие 6 женщин и 14 мужчин.

Были произведены несколько записей голоса каждого пациента (длительные гласные, цифры, слова и короткие предложения). Каждая запись характеризуется двадцатью шестью показателями:

- Jitter: local — средняя абсолютная разница между последовательными периодами звуковой волны, деленная на средний период;
- Jitter: local, absolute — средняя абсолютная разница между последовательными периодами звуковой волны в секунду;
- Jitter: rap — средняя абсолютная разница между периодом звуковой волны и средним среди него и двух соседних периодов;
- Jitter: ppq5 — средняя абсолютная разница между периодом звуковой волны и средним среди него и четырех соседних периодов;
- Jitter: ddp — средняя абсолютная разность между последовательными изменениями в периодах звуковой волны, деленная на средний период.

- Shimmer: local — средняя абсолютная разница между амплитудой последовательных участков записи, деленная на среднюю амплитуду;
- Shimmer: local, dB — средний абсолютный десятичный логарифм разности между амплитудами последовательных участков записи, умноженный на 20;
- Shimmer: apq3 — средняя абсолютная разница между амплитудой участка записи и средней амплитудой его и двух соседних участков, деленная на среднюю амплитуду;
- Shimmer: apq5 — средняя абсолютная разница между амплитудой участка записи и средней амплитудой его и четырех соседних участков, деленная на среднюю амплитуду;
- Shimmer: apq11 — средняя абсолютная разница между амплитудой участка записи и средней амплитудой его и десяти соседних участков, деленная на среднюю амплитуду;
- Shimmer: ddp — средняя абсолютная разница между последовательными изменениями амплитуд подряд идущих участков записи;
- AC — разница в уровне громкости, выражающаяся в децибелах;
- NTH (Noise-to-Harmonics) и HTN (Harmonics-to-Noise) — коэффициент меры охриплости, Harmonics и Noise выражаются в децибелах;
- медиана основного тона;
- средняя высота;
- стандартное отклонение;
- минимальная высота;
- максимальная высота;
- число импульсов;
- число периодов;
- средний период;

- стандартное отклонение периода;
- доля локально не голосовых кадров;
- количество разрывов речи;
- степень разрыва речи.

Глава 2. Обзор используемых алгоритмов

2.1. Метод опорных векторов

В задаче двуклассового обучения цель метода опорных векторов (SVM — support vector machine) — найти наилучшую функцию, разделяющую обучающие данные на два класса. Для линейно разделяемого набора данных линейная функция классификации соответствует разделяющей гиперплоскости $f(x)$, которая проходит через середину двух классов, разбивая их. Когда функция построена, принадлежность новых данных к тому или иному классу определяется ее знаком.

Поскольку существует бесконечно много подобных линейных функций, задача SVM состоит в максимизации отступа между элементами каждого класса и самой гиперплоскости. Геометрически отступ можно определить, как пространство, которое ограничено расстоянием между двумя ближайшими элементами, принадлежащими разным классам. Это определение позволяет нам выяснить, как максимизировать отступ таким образом, что даже при бесконечном количестве гиперплоскостей только некоторые будут являться решением МОВ. С целью убедиться, что найденное расстояние действительно максимально, классификатор пытается максимизировать функцию по аргументам $\overrightarrow{w}$ и b:

$$L_p = \frac{1}{2}||\overrightarrow{w}|| + \sum_{i=1}^{t} \alpha_i y_i (\overrightarrow{w} \cdot \overrightarrow{x_i} + b) + \sum_{i=1}^{t} \alpha_i,$$

где t – число тренировочных элементов, α_i — неотрицательные числа такие, что производная функции L_p по α_i равна нулю, при этом α_i — множитель Лагранжа, а L_p — лагранжиан. В этом равенстве вектор $\overrightarrow{w}$ и константа b определяют гиперплоскость. На рисунке 1 представлена диаграмма метода опорных векторов.

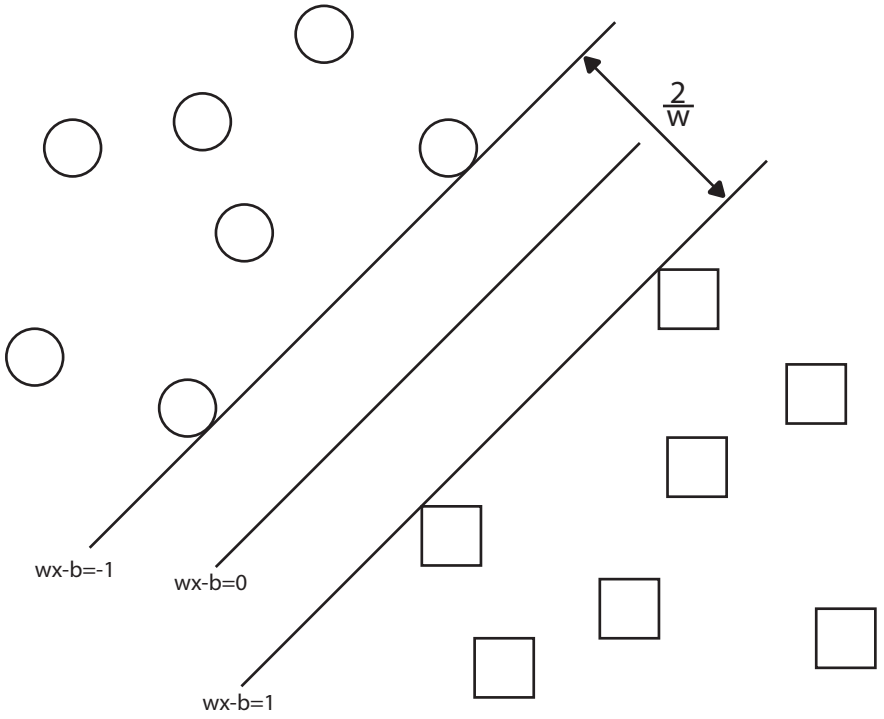

Рис. 1: Метод опорных векторов

Преимущество данного алгоритма в том, что он эффективен для многомерных пространств и даже в том случае, если размерность пространства больше количества экземпляров. Так как алгоритм размещает предсказываемый экземпляр выше или ниже гиперплоскости, нет вероятностной интерпретации принадлежности к тому или иному классу.

2.2. Дерево принятия решений

Алгоритм дерева принятия решений является методом машинного обучения с учителем. DTree (Decision tree — дерево решений) строит блок-схему, «задавая вопрос» относительно входящего атрибута, и получает ответ в виде двух или более результатов. Процесс построения ветви заканчивается, когда ответом на «вопрос» становится класс, к которому принадлежит элемент. Каждый лист представляет выходные данные, внутренние узлы соответствуют входным. На рисунке 2 представлено построенное дерево на примере принятия решения о выдаче кредита.

Рис. 2: Метод дерева принятия решений

Входные данные могут иметь количественный либо категориальный признак, и от этого зависит конечный результат условия. Для количественного показателя A они ограничиваются порогом h, который выбирается путем сортировки данных, максимизируя вероятность правильного предсказания принадлежности к классу. Для каждого дискретного значения атрибута A соответствует один результат, отличный от других, но метод позволяет сгруппировать исходящие данные в две и более группы с одним исходом для каждого подмножества.

Одним из недостатков данного алгоритма является переобучение. Чтобы решить эту проблему, используют метод “обрезки” ветвей дерева. Данный алгоритм убирает лишние ветви, основываясь на плохой оценке частоты ошибки для числа элементов, которые не относятся к настоящему

классу. Процесс отсечения ветвей движется от листьев к корню. В итоге мы получаем схему, которая с большой скоростью и вероятностью успеха предсказывает класс элемента.

Для того, чтобы с помощью данного алгоритма построить решающее дерево и применять его, данные должны удовлетворять нескольким условиям:

1. Информация об объектах, которые необходимо классифицировать, должна быть представлена в виде конечного набора признаков (атрибутов), каждый из которых имеет дискретное или числовое значение. Такой набор атрибутов назовем примером. Для всех примеров количество атрибутов и их состав должны быть постоянными;
2. Множество классов, к которым будут относиться примеры, должно быть конечным, а каждый экземпляр в обучающей выборке должен однозначно принадлежать к какому-либо определенному классу. Описываемый метод не подходит для классификации с нечеткой логикой, когда экземпляры принадлежат к классу с некоторой долей вероятности;
3. В обучающей выборке количество примеров должно быть значительно больше количества классов, и каждый экземпляр должен быть отнесен к своему классу.

2.3. Метод k наиближайших соседей

Метод k наиближайших соседей — это метрический, непараметрический классификатор. Непараметрическими классификаторами называют те, которые не делают никаких предположений относительно обучающей выборки. Его суть состоит в том, чтобы отнести экземпляр к тому классу, к которому принадлежат k его ближайших соседей. Обычно k — нечетное число, так как при четном может возникнуть неопределенность.

kNN (k-nearest neighbors) предполагает, что объекты находятся в метрическом пространстве. Данные могут быть либо скалярами, либо многомерными векторами. Поскольку точки находятся в пространстве признаков, они имеют понятие расстояния. Оно не обязательно должно быть евклидовым, хотя обычно используется именно оно. Каждый объект из обучающих данных состоит из набора векторов и метки класса. В простейшем, бинарном, случае результатом будет + или − (для положительных или отрицательных классов). Но kNN может одинаково хорошо работать с любым количеством классов. На рисунке 3 представлена геометрическая интерпретация метода k наиближайших соседей при $k = 3$ для бинарной классификации.

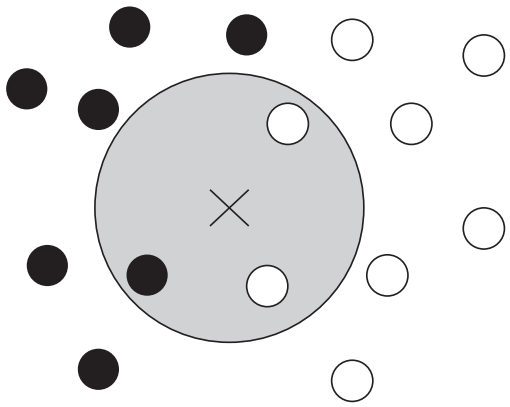

Рис. 3: Метод k наиближайших соседей

2.4. Многослойная нейронная сеть

Многослойная нейронная сеть основывается на упрощенном представлении нейронных связей человеческого мозга. На рисунке 4 изображена структура многослойной нейронной сети.

Входные нейроны — атрибуты экземпляров. Выходной нейрон — булева константа, определяющая принадлежность к классу. Значимость нейрона l-го слоя для нейрона $(l + 1)$-го слоя определяется коэффициентом, называемым весом. Чтобы найти значение нейронов скрытых и выходно-

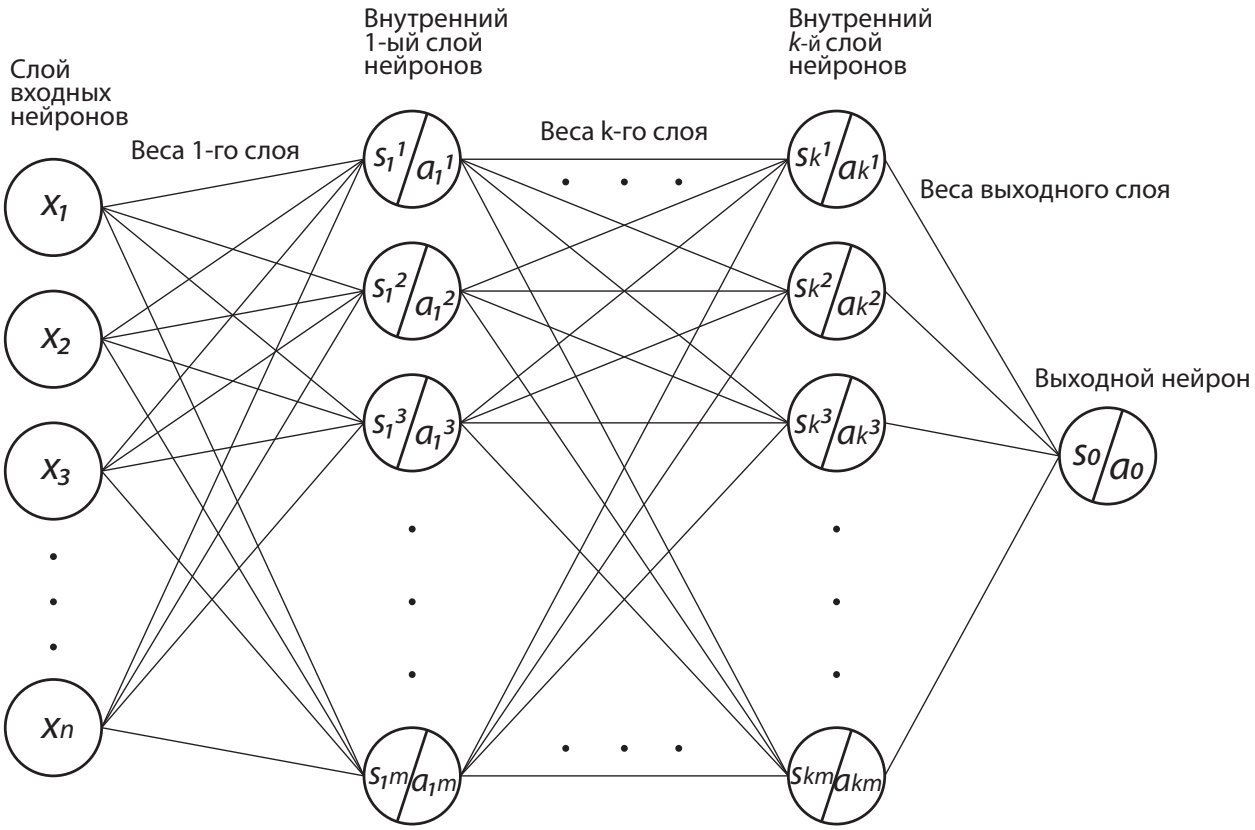

Рис. 4: Многослойная нейронная сеть

го слоев, вычисляется сумматорная функция, равная сумме произведений значений нейронов предыдущего слоя на соответствующие им веса:

$$s_l^j = \sum_{i=1}^{n} s_{l-1}^i \cdot w_l^{ij},$$

где s_l^j — j-й нейрон l-го слоя, для которого вычисляется сумматорная функция; w_l^{ij} — вес, соответствующий j-му нейрону, исходящий из i-го нейрона $(l-1)$-го слоя.

Окончательный результат получается, когда сумматорная функция проходит через целевую функцию, которая для разных задач отлична. В нашем случае, в задаче двуклассового прогнозирования, таковой была выбрана сигмоида:

$$\sigma(s) = \frac{1}{1+e^{-s}}.$$

Цель алгоритма — минимизация функции ошибки

$$J = \frac{1}{2}(o - y)^2,$$

где o — значение выходного нейрона, y — истинный результат для экземпляра обучающей выборки. Взяв частные производные этой функции по весам, можно определить, насколько должны измениться эти веса для получения минимальной ошибки. Удобно представлять веса и значения нейронов в матричном виде. Ниже представлены формулы, необходимые для нахождения разницы весов.

$$\delta^L = \nabla_a J \odot \sigma'(z^L),$$

$$\delta^l = \left(\left(W^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'(z^l),$$

$$\frac{\partial J}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j.$$

2.5. Скользящий контроль

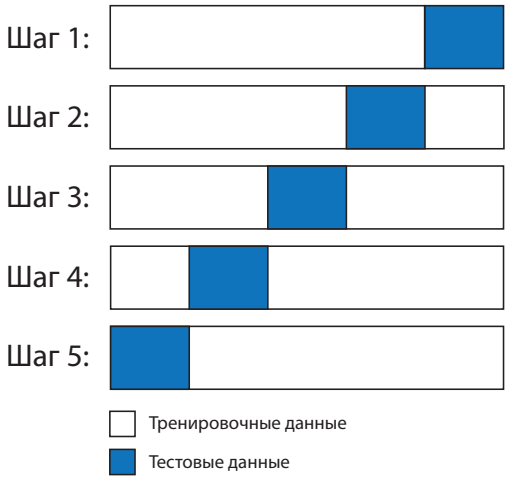

Рис. 5: Скользящий контроль

Для более объективной проверки эффективности работы алгоритма используют метод скользящего контроля (cross validation). Его суть состоит в том, что выборку данных делят на n непересекающихся частей, после

этого модель обучают на $n - 1$ частях, а на одной, оставшейся, части тестируют. На каждом из n этапов тестируемая часть разная. В настоящем исследовании $n = 5$. На рисунке 5 изображена диаграмма пятиэтапного скользящего контроля.

Глава 3. Сравнительный анализ эффективности алгоритмов

Для скользящего контроля данные анализа голоса и речи, взятые из открытого репозитория данных для машинного обучения UCI, были разделены на пять неперсекающихся подмножеств. Каждый метод машинного обучения был обучен и протестирован на этих данных. Всего экземпляров в наборе данных 1040.

Будем обозначать:

1. tp — сумма истинно-положительных решений;
2. tn — сумма истинно-отрицательных решений;
3. fp — сумма ложно-положительных решений;
4. fn — сумма ложно-отрицательных решений.

В таблице 1 представлены результаты работы алгоритмов.

Таблица 1

Алгоритм	**tp**	**tn**	**fp**	**fn**
SVM	353	341	179	167
DTree	302	334	186	218
kNN	327	363	157	193
MLP	425	380	95	140

Из результатов работы алгоритмов видно, что наибольшее количество правильных ответов дает многослойная нейронная сеть.

Интересно также сравнить эффективность методов с помощью метрик оценки качества классификации. В данном исследовании сравнительный анализ эффективности алгоритмов проводился с помощью следующих метрик:

1. Accuracy — доля правильных ответов, предсказанных алгоритмом, среди всех тестовых данных;

2. Precision — доля объектов, действительно принадлежащих данному классу относительно всех объектов, которые система отнесла к этому классу;
3. Recall — доля найденных классификатором объектов, принадлежащих классу относительно всех объектов этого класса в тестовой выборке;
4. F-measure представляет собой гармоническое среднее между Precision и Recall.

В таблице 2 представлены результаты оценок, основанных на вышеописанных метриках, для каждого алгоритма.

Таблица 2

Алгоритм	**Accuracy**	**Precision**	**Recall**	**F-measure**
SVM	0,67	0,66	0,68	0,67
DTree	0,61	0,62	0,58	0,59
kNN	0,66	0,67	0,63	0,65
MLP	0,77	0,82	0,75	0,78

Из таблицы видно, что многослойная нейронная сеть опережает другие методы по всем показателям. Поэтому именно этот алгоритм был выбран в качестве классификатора при разработке системы автоматизированной медицинской диагностики.

Глава 4. Разработка веб-приложения

4.1. Обзор используемых технологий

4.1.1. Spring framework

Spring framework предоставляет исчерпывающий инструмент для программирования и конфигурации современных приложений на Java независимо от платформы развертывания. Ключевым элементом Spring является инфраструктурная поддержка на прикладном уровне: Spring был разработан для того, чтобы разработчики могли сосредоточиться на бизнес-логике приложения, не тратя время на написание "закулисного"кода. Spring помогает решить многие задачи, с которыми сталкиваются Java-разработчики и организации, которые хотят создать информационную систему, основанную на платформе Java. Из-за широкой функциональности трудно определить наиболее значимые структурные элементы, из которых состоит этот фреймворк.

Основное преимущество Spring — возможность разработки приложения как набора слабосвязанных (loose-coupled) компонентов. Чем меньше компоненты приложения знают друг о друге, тем проще разрабатывать новый и поддерживать существующий функционал приложения. Классический пример — управление транзакциями. Spring позволяет вам управлять транзакциями совершенно независимо от основной логики взаимодействия с БД. Изменение этой логики не порушит транзакционность, равно как изменение логики управления транзакциями не сломает структуру программы. Spring поощряет модульность. Компоненты можно добавлять и удалять почти независимо друг от друга. Также Spring заметно упрощает модульное тестирование: в модуль, разработанный для работы в IoC контейнере, очень легко вставлять искусственные зависимости и проверить работу только этого компонента. Spring сильно облегчает инициализацию и

конфигурацию компонентов программы, позволяя гибко настраивать приложение без существенных изменений Java-кода.

Конфигурирование приложения в Spring можно осуществлять как с помощью xml-файлов, так и с помощью аннотаций. Также, ничего не мешает комбинировать оба подхода. Наиболее традиционный, но сложный путь, который используется в фреймворке с первой версии, — построение xml-зависимостей. С введением аннотаций в языке Java 5 появилась возможность настраивать фреймворк с помощью них. В разработке данного приложения использовались аннотации.

4.1.2. Hibernate

В первую очередь необходмо рассказать об ORM (Object-relational mapping). ORM — это отображение объектов реляционных баз данных в классы какого-либо объектно-ориентированного языка со всеми их полями, значениями и отношениями между друг другом.

ORM-решением для языка программирования Java, является технология Hibernate, которая не только устанавливает связи Java классов с таблицами базы данных (и типов данных Java в типы данных SQL), но также предоставляет возможность для автоматической генерации запросов и извлечения данных и может заметно уменьшить время, которое обычно тратится на ручное написание SQL и JDBC кода. Hibernate создает SQL вызовы и освобождает программиста от ручной обработки финального набора данных и конвертации объектов, сохраняя приложение портативным во все SQL базы данных.

Для решения своих задач Hibernate должен получить от разработчика следующую информацию:

1. параметры соединения с базой данных;
2. описание отображения классов на таблицы. Данное описание вклю-

чает в себя связь между колонкой в таблице и полем в классе;

3. описание отношений между классами.

4.1.3. PostgreSQL

PostgreSQL — мощная система управления объектно-реляционными базами данных с открытым исходным кодом. Она работает на всех основных операционных системах, включая Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, macOS, Solaris, Tru64) и Windows. Она полностью отвечает правилам ACID (Atomicity — Атомарность, Consistency — Согласованность, Isolation — Изолированность, Durability — Долговечность), имеет полную поддержку внешних ключей, объединений, представлений, триггеров и хранимых процедур (на нескольких языках). СУБД включает в себя большинство типов данных SQL. Она также поддерживает хранение больших объектов, включая изображения, звуки или видео. Она имеет собственные интерфейсы программирования для C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC.

Версия корпоративного уровня PostgreSQL имеет такие сложные функции, как Multi-Version Concurrency Control (MVCC), восстановление времени, табличные пространства, асинхронная репликация, вложенные транзакции (точки сохранения), онлайн/горячие резервные копии, сложный планировщик запросов и запись для обеспечения отказоустойчивости. Она поддерживает международные наборы символов, многобайтовые кодировки символов, Unicode. Она хорошо масштабируется как при огромном количестве данных, которым она может управлять, так и с большим количеством одновременно работающих пользователей. В производственных средах работают активные PostgreSQL-системы, которые обрабатывают более 4 терабайт данных.

4.1.4. Maven

Maven — фреймворк для автоматизации большого количества задач при создании и поддержке программы. Не зависит от платформы и языка программирования. Функциональность Maven можно условно разделить на три части: конфигурация модели проекта (POM), репозиторий артефактов, управление жизненным циклом проекта.

POM (Project object model) — конфигурационный файл проекта, в котором описаны компоненты проекта и устройство его жизненного цикла. В этом файле описываются общие характеристики проекта: имя, версия, авторы и их контактная информация, система управления версиями проекта и связанные с ней сетевые ресурсы, тип проекта, связи с другими проектами, плагины, используемые при сборке, и описание способа их применения.

Репозитории артефактов — хранилища результатов сборки проекта, устроенные так, чтобы облегчить поиск нужного артефакта (по имени, версии, типу). Они позволяют разработчикам вести командную работу без необходимости хранить копии исходных кодов их модулей. Также становится ненужным выполнять сборку дерева зависимостей с нуля. Стоит обратить внимание, что POM-файл, содержащий конфигурацию проекта, также является артефактом и может храниться в репозиториях вместе со всеми. Таким образом репозиторий является в том числе источником информации о хранимых в нем модулях.

Жизненным циклом называется множество операций, выполняемых над проектом: от инициализации сборки до развертывания. Жизненный цикл разделен на фазы:

1. validate — проверяет корректность метаинформации о проекте;
2. compile — компилирует исходный код;
3. test — тестирует код из предыдущего шага;

4. package — упаковывает скомпилированые классы в формат .jar или .war;
5. integration-test — отправляет упаковынные классы в среду интеграционного тестирования и тестирует их;
6. verify — проверяет корректность пакета и удовлетворение требованиям качества;
7. install — устанавливает пакет в локальный репозиторий, откуда он (пакет) будет доступен для использования как зависимость в других проектах;
8. deploy — отправляет пакет на удаленный production сервер, откуда другие разработчики его могут получить и использовать.

Каждая фаза переводит модуль в новое состояние. Каждая предыдущая фаза подготавливает основу для следующей.

4.1.5. Tomcat

Apache Tomcat — современный сервер приложений, обеспечивающий работу проектов на языке Java. Tomcat является отдельным веб-сервером, который обеспечивает работу как динамической части сайта, так и статических файлов (изображения и т.д.). Tomcat написан на Java, ресурсоемок, и желательно снять с него нагрузку по работе с медленными конечными клиентами.

4.1.6. Heroku

Heroku — облачная платформа для развертывания сайтов и веб-приложений. Поддерживает различные языки программирования (Java, PHP, Ruby, Scala, Go). Также сервис Heroku предоставляет платформу, основанную на PostgreSQL, обеспечивающую безопасную масштабируемую базу данных с множеством инструментов для разработчиков.

4.2. Архитектура приложения

4.2.1. Требования к архитектуре

Проектируя приложение, программист должен уделять большое внимание архитектуре, так как плохо продуманная структура проекта может повлечь за собой трудности при разработке продукта. Программу с хорошей архитектурой легче масштабировать, изменять, а также тестировать. Ниже представлены основные критерии хорошей архитектуры.

Эффективность системы. Разрабатываемый продукт должен отвечать поставленным задачам и правильно выполнять свои функции, причем в различных условиях. Сюда можно отнести такие характеристики, как надежность, безопасность, производительность, способность справляться с увеличением нагрузки и т.п.

Гибкость системы. Каждый проект по различным причинам со временем требует каких-либо изменений — изменяется функционал, добавляются новые возможности в приложение. Чем быстрее и удобнее можно внести изменения в существующую программу, чем меньше проблем и ошибок это вызовет, тем гибче система. Изменение одного модуля программы не должно влиять на другие ее компоненты.

Расширяемость системы. Возможность обогащать систему новыми сущностями и функциями, не нарушая ее основной структуры, — важнейшее свойство архитектуры. На начальном этапе в систему имеет смысл закладывать лишь основной и самый необходимый функционал. Но при этом архитектура должна позволять легко наращивать дополнительный функционал по мере необходимости. Причем так, чтобы внесение наиболее вероятных изменении требовало наименьших усилий.

Требование, чтобы архитектура проекта была гибкой и расширяемой, является настолько важным, что оно даже сформулировано в виде отдельного принципа — «Принцип открытости/закрытости» (Open-Closed

Principle). Программные сущности (классы, модули, функции и т.п.) должны быть открытыми для расширения, но закрытыми для модификации. Иначе говоря, должна быть возможность расширять и изменять функционал системы без изменения или переписывания уже существующих компонентов. Именно этот принцип является основой «плагинной архитектуры» (Plugin Architecture).

Масштабируемость процесса разработки. Архитектура должна позволять распределить процесс разработки между несколькими программистами, так чтобы множество людей могли работать над созданием программы одновременно.

Тестируемость. Код, который легче тестировать, будет содержать меньше ошибок и обладать большей надежностью. Но тесты не только улучшают качество кода — многие разработчики приходят к выводу, что соблюдение этого критерия непосредственным образом ведет к хорошему дизайну, и одновременно является одним из важнейших характеристик, позволяющих оценить его качество.

Стуктурированность, читабельность, сопровождаемость. В процессе проектирования и разработки часто принимает участие большое количество людей. После релиза продукта сопровождать программу, возможно, придется людям, не участвовавшим в ее разработке. Поэтому хорошая архитектура должна предоставлять новым программистам возможность легко и в короткие сроки разобраться в проекте. Программа должна быть хорошо структурирована, не содержать дублирования, иметь хорошо оформленный код и желательно документацию.

4.2.2. Структура таблиц базы данных

Прежде чем создавать таблицы базы данных нужно четко понимать, какие сущности будут присутствовать в проекте, какие атрибуты будут соответствовать каждой из них и какие действия пользователь будет совер-

шать над ними. Очень важно тщательно продумать структуру, так как ее дальнейшее исправление может повлечь возникновение ошибок и ненужный рефакторинг кода, который отнимет много времени. Ниже описана функциональность веб-приложения.

Пройдя процедуру аутентификации, пользователь — лаборант, принимающий анализы у пациентов, — может выпонить следующие действия: просмотреть результаты проведенных ранее анализов, принять новый анализ у пациента, добавить новый анализ в базу (обучить нейронную сеть выносить результат по новому анализу).

Таким образом в базе данных должны храниться модели доктора (он же лаборант, doctor), пациента (patient) и анализа (веса для классификации нейронной сетью, analysis). Также необходима таблица, в которой будут храниться анализы, принятые лаборантом у пациента, (labwork). В рамках данной работы модели будут обладать, возможно, не всеми бюрократически достаточными атрибутами, но всеми очевидно необходимыми.

Поля модели "doctor": doctor_id (serial), second_name (varchar), patronymic (varchar), first_name (varchar).

Поля модели "patient": patient_id (serial), second_name (varchar), patronymic (varchar), first_name (varchar), policy_number (long).

Поля модели "analysis": analysis_id (serial), title (varchar), key (varchar), weights (double precision[ ][ ][ ]), date (date), description (varchar).

Поля модели "labwork": labwork_id (serial), analysis_id (связан с analysis_id из модели "analysis", many-to-one), doctor_id (связан с doctor_id из модели "doctor", many-to-one), patient_id (связан с из patient_id модели "patient", many-to-one), result (double precision[ ]), description (varchar), appointment_date (date), having_date (date), _receiving_date (date). Ниже, на рисунке, представлена схема базы данных.

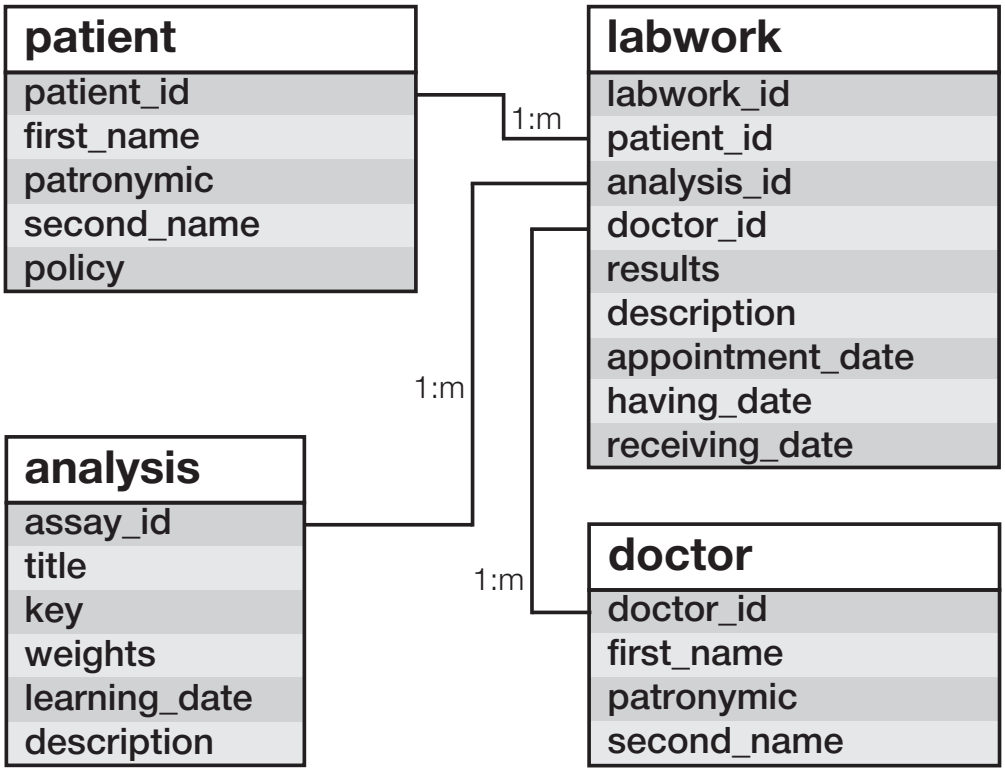

Рис. 6: Схема базы данных

4.2.3. Описание архитектуры проекта

В данном проекте использовалась четырехуровневая архитектура приложения. Эта структура была выбрана для того, чтобы уменьшить связность компонентов приложения. Это позволит придерживаться вышеописанных критериев хорошей архитектуры. На рисунке 7 изображена архитектура проекта, разрабатываемого в рамках данной выпускной квалификационной работы.

Первый уровень — Model — служит для связи объектов приложения с сущностями таблиц базы данных. Полями класса являются столбцы таблицы базы данных. Связь осуществляется посредством аннотаций @Entity и @Table из библиотеки *javax.persistence*. Model выполняет команды второго уровня (Repository), изменяя состояние объектов.

Следующий слой — Repository — необходим для управления и изменения объектов базы данных. С помощью средств Hibernate генерирует SQL-запросы и создает, редактирует или удаляет объекты базы данных.

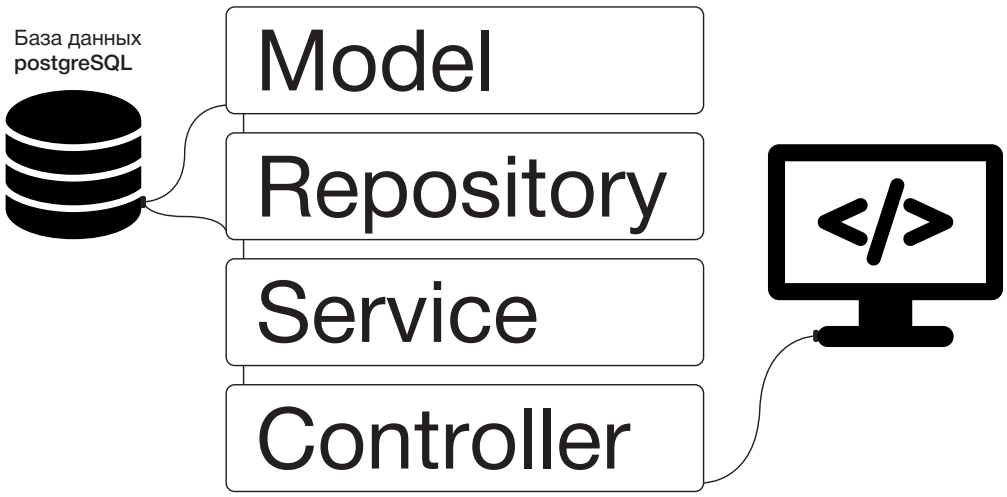

Рис. 7: Архитектура проекта

Транзакции создаются при помощи библиотеки *org.hibernate.Session*. Для того, чтобы Spring framework воспринимал этот класс, как репозиторий, необходимо перед его объявлением поставить аннотацию @Repository.

Третий уровень — Service — является соединительным звеном между репозиторием и контроллером. Класс Service проверяет и выполняет бизнес-логику приложения. Данный модуль собирает данные с контроллера и обращается к репозиторию для управления данными. Для корректной работы в контексте Spring framework перед объявлением класса следует добавить аннотацию @Service("some_name"), которая находится в классе *org.springframework.stereotype.Service*.

Финальный слой — Controller. Именно с его помощью происходит взаимодействие между сервером и клиентом. Этот класс предоставляет готовые данные для front-end разработки и наоборот получает информацию от клиента. В нем концентрируется вся бизнес-логика проекта. Необходимо добавить аннотацию @Controller, располагающуюся в классе *org.springframework.stereotype.Controller*.

Заключение

В результате проделанной работы была спроектирована и разработана система автоматизировнной медицинской диагностики. Для достижения этой цели был проведен сравнительный анализ эффективности метода опорных веторов, метода дерева решений, метода k наиближайших соседей и многослойной нейронной сети. По результатам анализа, наилучший показатель среднего гармонического Precision и Recall был у многослойной нейронной сети. На ее основе и было разработано веб-приложение. Демоверсию программы можно протестировать на сайте:

https://smartmed2017.herokuapp.com.

На данный момент приложение обладает набором функций для лаборанта, принимающего анализы у пациентов. Лаборант может узнать диагноз, загрузив в систему результаты анализа, просмотреть результаты уже проведенных анализов, добавить данные для нового, еще не зарегистрированного в системе вида анализа.

В ходе дальнейшей работы планируется добавить новые модели пользователей: доктор и пациент; расширить для них функционал приложения. Улучшить работу нейронной сети: эмпирическим путем — с помощью скользящего контроля — подобрать наилучшее количество скрытых слоев и нейронов внутри них.

Список литературы

[1] Scikit-learn http://scikit-learn.org/stable/

[2] Wu X., Kumar V. Top 10 algorithms in data mining // Knowl Inf Syst, 2008. Vol. 14(1). P. 1–37

[3] Bishop C. M. Neural Networks for Pattern Recognition. 2 ed. Oxford: Oxford University Press, 1995. 482 p.

[4] Madjarov G., Kocev D. An extensive experimental comparison of methods for multi-label learning // Pattern Recognition, 2012. Vol. 45, P. 3084–3104.

[5] Lewis D. D. Evaluating Text Categorization // Computer and Information Science Dept., 1991. P. 312–318.

[6] Geisser S. Predictive Inference. Predictive Inference. New York: Chapman and Hall, 1993. 240 p.

[7] Fowler M. Patterns of Enterprise Application Architecture. 1 ed. Boston: Addison-Wesley, 2003. 533 p.

[8] Хо К., Харроп Р. Spring 3 для профессионалов. М.: Вильямс, 2013. 880 с.

[9] Bauer C., King G. Java Persistence with Hibernate. Greenwich: Manning, 2005. 841 p.

[10] Erdogdu Sakar, B., Isenkul, M.Collection and Analysis of a Parkinson Speech Dataset with Multiple Types of Sound Recordings // IEEE Journal of Biomedical and Health Informatics, 2013. Vol. 17(4). P. 828–834.

[11] UCI — Machine Learning Repository https://archive.ics.uci.edu